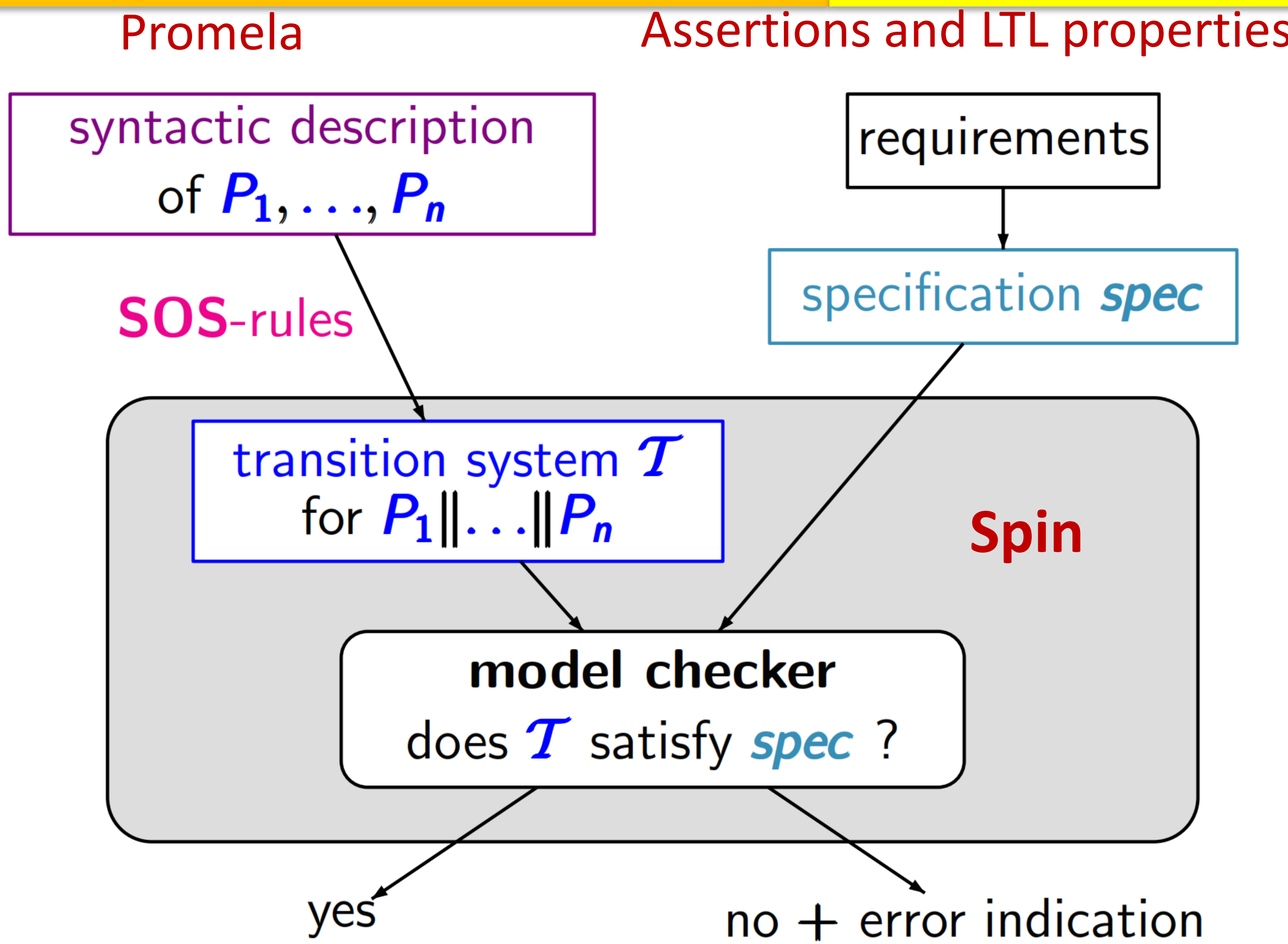
Model Checking with Spin

Fatemeh Ghassemi
Hossein Hojjat
University of Tehran

A big picture



Spin

Spin is a popular open-source software verification tool, used by thousands of people worldwide. The tool can be used for the formal verification of multi-threaded software applications. The tool was developed at Bell Labs in the Unix group of the Computing Sciences Research Center, starting in 1980. The software has been available freely since 1991, and continues to evolve to keep pace with new developments. In April 2002 the tool was awarded the ACM System Software Award. [read more]

discover

- what is spin?
- success stories
- examples
- roots

learn

- tutorials
- books
- papers
- model extraction
- exercises

use

- installation
- man pages
- options
- releases

community

- forum
- symposia
- support
- projects

Open Source: Starting with Version 6.4.5 from January 2016, the Spin sources are available under the standard BSD 3-Clause open source license. Spin is now also part of the latest stable release of Debian Linux, and has made it into the 16.10+ distributions of Ubuntu. The current Spin version is 6.4.7 (August 2017).

Visit at http://spinroot.com/spin/whatispin.html

foundations

1936 1950 1968 1975 1980 1989 1995 2000 2004 2016

Fortran Algol Algol68 C C++

1936: first theory on computability / Turing machines

1960: early work on ω-automata theory,

1940-50: first computers

1968: two terms introduced:
software crisis
software engineering

1955: early work on tense logics (predecessors of LTL)

key theoretical developments related to automata-theoretic verification

Taken from lecture 4 of **Logic Model Checking for Formal Software Verification** course at https://piazza.com/caltech/winter2017/cs118/resources

foundations

1936 1950 1968 1975 1980 1989 1995 2000 2004 2016

Fortran Algol Algol68 C

Pan C++ LTL CTL Spin SMV nuSMV Spin 6.0

1936: first theory on computability, e.g., Turing machines

1960: early work on ω-automata theory,

1955: early work on tense logics (predecessors of LTL)

1975: Edsger Dijkstra's paper on Guarded Command Languages

1978: Tony Hoare's paper on Communicating Sequential Processes

1977: Amir Pnueli introduces linear temporal logic for distributed system verification

1976-1979: first experiments with reachability analyzers (e.g., Jan Hajek: 'Approver')

1980: first version of Spin: 'Pan' (Bell Labs)

1981: Ed Clarke and Allen Emerson introduce the term 'model checking' and the logic CTL*

1986: Pierre Wolper and Moshe Vardi define the automata theoretic framework for LTL model checking

1986: Mazurkiewicz paper on trace theory

1989: Spin 1.0 released publicly verification of ω-regular properties

1993: BDDs and the SMV model checker (Ken McMillan, CMU)

1995: partial order reduction added to Spin. LTL conversion added to Spin. (Doron Peled)

2001: support for embedded C code in Spin version 4.0

2007-17: multi-core & swarm verification algorithms added in Spin 5.0-6.0

the two most used logic model checking systems:

Spin: explicit state LTL model checking based on automata theoretic verification targeting *concurrent software* verification

SMV: symbolic CTL model checking targeting *hardware* circuit verification

(there are many other types of model checkers)

Taken from lecture 4 of **Logic Model Checking for Formal Software Verification** course at https://piazza.com/caltech/winter2017/cs118/resources

Outline

- Introduction to Promela and Spin
  - Lecture 3 of Logic Model Checking for Formal Software Verification course at Caltech https://piazza.com/caltech/winter2017/cs118/resources
- Converting (a subset) of Promela to program graphs
  - SOS rules

Guarded command language

- Promela is based on guarded command language, provided by Dijkstra
  - a high-level modeling language that contains features of imperative languages and nondeterministic choice

guarded command $g \Rightarrow stmt$ ← enabled if g is true

g	:	guard, i.e., Boolean condition on the program variables
$stmt$	:	statement

Guarded Command Language (GCL)

guarded command $g \Rightarrow \textit{stmt}$ ← enabled if g is true

repetitive command/loop:

DO :: $g \Rightarrow \textit{stmt}$ OD ← WHILE g DO $\textit{stmt}$ OD

conditional command:

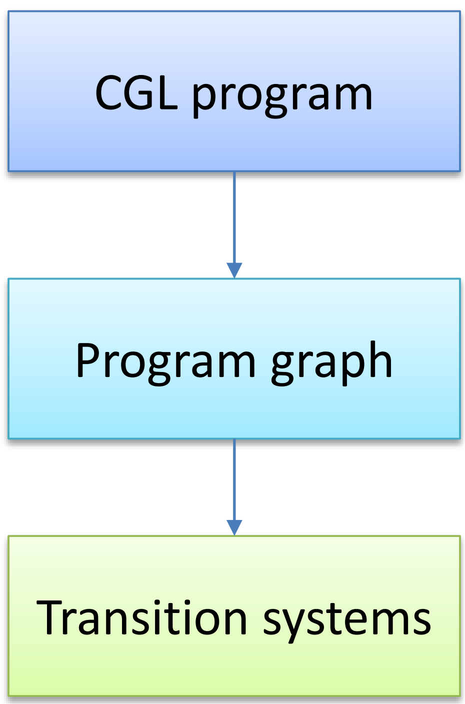
IF :: $g \Rightarrow \textit{stmt}_1$
:: $\neg g \Rightarrow \textit{stmt}_2$
FI

← IF g THEN $\textit{stmt}_1$ ELSE $\textit{stmt}_2$ FI

symbol :: stands for the nondeterministic choice between enabled guarded commands

Guarded command language (Con.)

- Semantics of CGL is given in terms of program graph Example : a program graph to an LTS

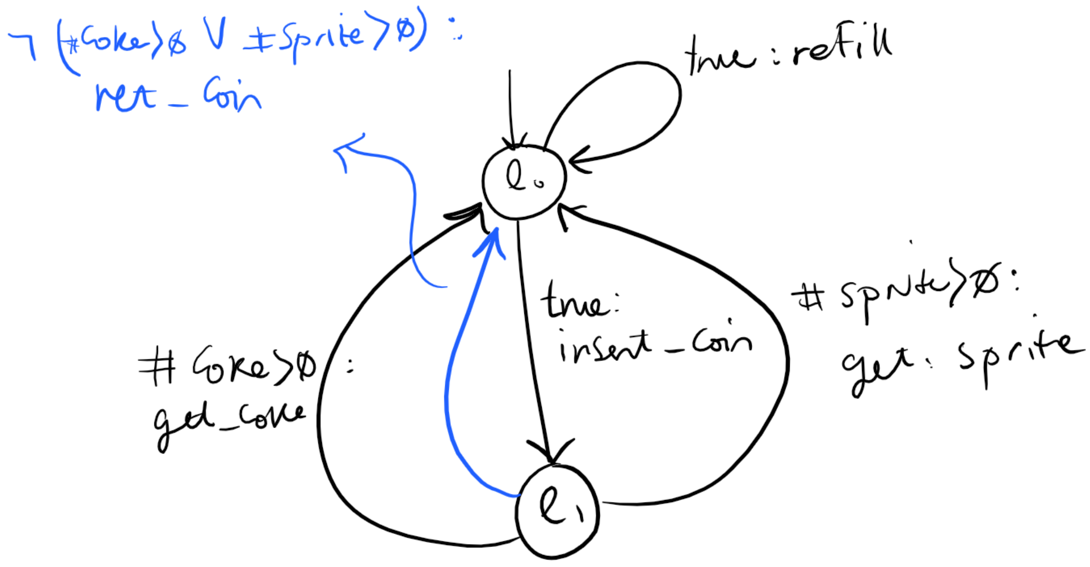


?

$$\frac{\ell \xhookrightarrow{g:\alpha} \ell' \;\wedge\; \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\eta, \alpha) \rangle}$$

Example: Beverage Machine

– Assume η=[#coke=1, #sprite=0]



$$\frac{l_1 \text{ -- \#coke>0:get\_coke ---> } l_0 + \eta \Vdash \text{\#coke>0}}{\langle l_1,\eta\rangle \text{ --- get\_coke ---------> } \langle l_0,[\text{\#coke=0, \#sprite=0}]\rangle}$$

Syntax of Promela

- A subpart of the syntax is given by the grammar below:

$$\begin{aligned}
\mathsf{stmt} \quad ::= \quad & \mathtt{skip} \mid x := \mathsf{expr} \mid c?x \mid c!\mathsf{expr} \mid \\
& \mathsf{stmt}_1 \,;\, \mathsf{stmt}_2 \mid \mathtt{atomic}\{\mathsf{assignments}\} \mid \\
& \mathbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \mathbf{fi} \quad \mid \\
& \mathbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \mathbf{do}
\end{aligned}$$

Semantics of Promela

- For x=expr where $x \in Var$

$$\frac{}{\text{skip --- true:id --- > exit}} \; : \text{skip}$$

$$\frac{}{\text{x=expr ---true: assign(x,expr) --- > exit}} \; : \text{expr}$$

Semantics of Promela (Con.)

- For send and receive on channels

―――――――――――――――――――――――――― : rec

c?x--- c?x --- > exit

――――――――――――――――――――――――――――――― : snd

c!expr---c!expr --- > exit

Semantics of Promela (Con.)

- For atomic region

$$\frac{}{\text{atomic}\{x_1=\text{expr}_1;\ldots x_m=\text{expr}_m\}\text{--true:}\alpha_m \text{ --- > exit}} \quad \text{: atomic}$$

where α_i=Effect(assign(x_i,expr_i),Effect(α_{i-1},η))

Semantics of Promela(Con.)

- For sequential composition $stmt_1;stmt_2$

$$\frac{stmt_1 \text{ --- } g : a \text{ ---> } stmt_1' \neq exit}{stmt_1 ;stmt_2 \text{ --- } g:a \text{ ---> } stmt_1';stmt_2} : Seq_1$$

$$\frac{stmt_1 \text{ --- } g : a \text{ ---> } exit}{stmt_1 ;stmt_2 \text{ --- } g:a \text{ ---> } stmt_2} : Seq_2$$

Semantics of Promela(Con.)

- For if-statement Test-and-Set semantics

$$\frac{stmt_i \text{ --- } g : a \text{ ---> } stmt'}{\text{if} :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \text{ fi --- } g \wedge g_i : a \text{ ---> } stmt'} \quad : \text{IF}$$

  – Note : the semantics of IF is given in terms of semantics of its elements Stm_1, …, Stm_n, i.e., its structure

Semantics of Promela (Con.)

- Example: derive its program graph

```
active proctype loop()
{    byte a, b;

                  if
                  :: a>1 -> b = 2*a;
                  :: b = 2*a; skip
                  fi;
}
```

Semantics of Promela (Con.)

- For do-statement Test-and-Set semantics

$$\frac{stmt_i \text{ --- } g : a \text{ ---> } stmt' \neq exit}{do :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \; od \text{ --- } g \wedge g_i : a \text{ ---> } stmt'; do :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \; od} : DO_1$$

$$\frac{}{do :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \; od \text{ --- } \neg g_1 \wedge \ldots \wedge \neg g_n \text{ ---> } exit} : DO_2$$

$$\frac{stmt_i \text{ --- } g : a \text{ ---> } exit}{do :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \; od \text{ --- } g \wedge g_i : a \text{ ---> } do :: g_1 \Rightarrow stmt_1 \ldots :: g_n \Rightarrow stmt_n \; od} : DO_3$$

Semantics of Promela (Con.)

- Example: derive its program graph

```
active proctype loop()
{   byte a, b;

    do
    :: a = (a+1)%3;
            if
            :: a>1 -> b = 2*a;
            :: b = 2*a; skip
            fi;
            b--
    od
}
```

Semantics of Promela (Con.)

- For if-statement Two-step semantics

$$\frac{}{\text{if} :: g_1 \Rightarrow \text{stmt}_1 \ldots :: g_n \Rightarrow \text{stmt}_n \text{ fi} \text{ --- } g_i\text{:id} \text{ ---> } \text{stmt}_i} : \text{IF}$$

Semantics of Promela (Con.)

- For do-statement Two-step semantics

$$\frac{}{\text{do} :: g_1 \Rightarrow \text{stmt}_1 \ldots :: g_n \Rightarrow \text{stmt}_n \text{ od --- } g_i\text{:id ---> stmt}_i\text{; do} :: g_1 \Rightarrow \text{stmt}_1 \ldots :: g_n \Rightarrow \text{stmt}_n \text{ od}} \; : \text{DO}$$